the group has a unique combination of the software components in the search space. The group of software-stack candidates may be a relatively small subset of the total number of software-stack combinations that can be generated using all of the software components in the search space. For example, the group may include 100 thousand software-stack candidates, which may be a relatively small subset of the 10 million total possibilities. In this way, the system has significantly reduced the amount of software-stack candidates that will undergo further analysis.

[0010] During or after the search process, the system can also determine respective scores for the software-stack candidates using a scoring function. In some examples, the scoring function can take into account hardware and software characteristics of a computing environment in which the target software item is to execute. Additionally or alternatively, the scoring function can take into account security properties of the software item and each of its dependencies. Additionally or alternatively, the scoring function can take into account performance properties of the software item and each of its dependencies. The scoring function may also take into account other factors. Using a scoring function to characterize each of the software-stack candidates in this way can be faster and less resource-intensive than building and testing each of the software-stack candidates to verify application behavior.

[0011] After scoring the software-stack candidates, the system can select one of the software-stack candidates from the group as a recommended software-stack based on its corresponding score. For example, the system can select whichever software-stack candidate in the group has the highest score or the lowest score, depending on the scoring function, as the recommended software-stack. The recommended software-stack may be the best software stack for a given computing environment relative to the rest of the software-stack candidates in the group. The system may then transmit an output indicating the recommended software-stack to a user, who may choose to install the recommended software-stack on a computing device.

[0012] In some examples, the user can provide input to the system indicating a target software item. The input can also include one or more characteristics of a computing environment in which the target software item will be executed. The system can use those characteristics to determine a respective score for each of the software-stack candidates, since the scoring function may take such characteristics into account (among other factors). The system can then select the recommended software-stack using one or more processes described herein. In this way, the system can determine a recommended software-stack that is the best for the computing environment's specific characteristics, relative to the other software-stack candidates. The system can then output the recommended software-stack, to enable the recommended software-stack to be included (e.g., installed) in the computing environment.

[0013] These illustrative examples are given to introduce the reader to the general subject matter discussed here and are not intended to limit the scope of the disclosed concepts. The following sections describe various additional features and examples with reference to the drawings in which like numerals indicate like elements but, like the illustrative examples, should not be used to limit the present disclosure.

[0014] FIG. **1** is a block diagram of an example of a system for determining a recommended software-stack for a software item according to some aspects of the present disclosure. The system **100** includes a client device **102**, such as a laptop computer, desktop computer, or mobile device (e.g., smartphone, tablet, or e-reader). The client device **102** can include a computing environment **104** for a target software item. Examples of the computing environment **104** can include a runtime environment or a build-time environment. The computing environment **104** may have certain characteristics, such as hardware characteristics **106** and software characteristics **108**. Examples of the hardware characteristics **106** can include the number and types of processors, non-volatile memory, and volatile-memory supporting the computing environment **104**. Examples of the software characteristics **108** can include the operating system and libraries in the computing environment **104**.

[0015] In some examples, the client device **102** can transmit a request to a server **112**. The request can serve as an input **110** indicating a target software item and a characteristic of the computing environment **104** in which the target software item is to be executed. The input **110** may not include a version indicator for the target software item, but may rather refer to the target software item more generally. The server **112** can receive the input **110** and responsively perform operations to determine a recommended software-stack **114** for the target software item. The recommended software-stack **114** can include a recommended version **116** of the target software item (denoted "TSI Version" in FIG. **1**) and one or more recommended dependencies **118** of the target software item, where such dependencies may be direct or indirect dependencies of the target software item. The server **112** can then generate an output **120** indicating the recommended software-stack **114** and provide the output **120** to the client device **102**. A user of the client device **102** may receive the output **120** (e.g., via a display of the client device **102**) and install the recommended software-stack **114**. Alternatively, the client device **102** can automatically install the recommended software-stack **114**, in some examples.

[0016] To determine the recommended software-stack **114**, the server **112** can begin by executing a search algorithm **122**. The search algorithm **122** can be a heuristic search algorithm, a stochastic search algorithm, or another type of search algorithm. The search algorithm **122** can be executed to recursively analyze direct and indirect dependencies of the target software item to develop a search space containing various combinations of the target software item and its dependencies. These combinations may include different versions of the target software item and different mixtures and versions of the dependencies. Each element in the search space can correspond to a unique combination of a specific version of the target software item and the dependencies of that version. The search space can then be searched based on an objective function. In some examples, the search algorithm **122** can learn over time (e.g., between iterations) to more rapidly converge towards a solution to the objective function, without having to analyze every element in the search space. This can reduce the practical magnitude of the search space, exponentially reducing the amount of computational time and resources required to perform the search. By performing the search, the server **112** can determine a group of software-stack candidates **124***a-n*, where each software-stack candidate **124***a-n* in the group corresponds to a unique combination of the target software item and its dependencies.